|  |
| --- |
| Homework : \_\_8A\_\_\_\_  //Christopher Badolato  //ENC 3211  //11/29/18  //Assignment 8  #include<conio.h>  #include<stdio.h>  #include<stdlib.h>  #include<math.h>  /\* borrowed from http://www.dailyfreecode.com/code/general-newton-raphson-method-2405.aspx  I edited the code to fit the output of the assignment  \*/  int power = 4, i=0, cnt=1, flag=0;  int coef[10]={0};  float x1=0, x2=0, t=0;  float fx1=0, fdx1=0;  void main()  {  //grabs our user entered values to apply Newton-Raphson method.  printf("Finding root for three degree polynomial Ax^3 + Bx^2 + Cx + D\n using Newton-Raphson method\n\n");  printf("Enter the value for the following coefficients\n\n");  printf("A:");  scanf("%d",&coef[0]);  printf("B:");  scanf("%d",&coef[1]);  printf("C: ");  scanf("%d",&coef[2]);  printf("D:");  scanf("%d",&coef[3]);  printf("\nInitial root estimate : ");  scanf("%f",&x1);  printf("\n %d : Estimate : \t%.10f ",cnt,x1);  //apply's Newton-Raphson Method and prints solution intervals  do  {  cnt++;  fx1=fdx1=0;  for(i = power; i >= 1; i--)  {  fx1 += coef[i] \* (pow(x1,i)) ;  }  fx1+=coef[0];  for(i = power; i >= 0; i--)  {  fdx1+=coef[i]\* (i\*pow(x1,(i-1)));  }  t=x2;  x2=(x1-(fx1/fdx1));  x1=x2;  printf("\n %d : Estimate : \t%.10f ",cnt,x2);  }while((fabs(t - x1))>=0.00001); // added another o used to be 0.0001  printf("\n\nApproxmiate solution %.10f",x2);  getch();  } |
|  |
| Finding root for three degree polynomial Ax^3 + Bx^2 + Cx + D |
| Homework : \_\_8B\_\_\_\_  //Christopher Badolato  //ENC 3211  //11/29/18  //Assignment 8  /\* This program computes root of function with bisector method \*/  #include <stdio.h>  #include <math.h>  /\* Borrowed from http://scalettar.physics.ucdavis.edu/cosmos/bisector.pdf  Changed to match our output with the intervals\*/  double myFunction(double x, double a, double b,double c, double d){  return a\*x\*x\*x + b\*x\*x + c\*x + d;  }  int main(void)  {  double a,b,c,d;  double leftpt, rightpt, midpt, epsilon = .0000000001;  double midvalue, rtvalue, root;  int count = 1;  //scans input from user  printf("Finding the root for three degree polynomial Ax^3 + Bx^2 + Cx + D \nusing Bisection method.\n");  printf("\nEnter values for the following coefficients\n");  printf("A :");  scanf("%lf", &a);  printf("B :");  scanf("%lf", &b);  printf("C :");  scanf("%lf", &c);  printf("D :");  scanf("%lf", &d);  printf("\nInitial Bracket for root :\n");  printf("Low value:");  scanf("%lf", &leftpt);  printf("High value:");  scanf("%lf", &rightpt);  printf("\n");  //applies bisection method to the given values with our polynomial  do {  midpt = (leftpt + rightpt)/2;  rtvalue = myFunction(rightpt,a,b,c,d);  midvalue = myFunction(midpt,a,b,c,d);  //print left point/ midpoint.  printf("%d :New interval: [ %15.10lf .. %15.10lf ]\n", count, leftpt, midpt);  if (rtvalue \* midvalue >= 0)  rightpt = midpt;  else leftpt = midpt;  root = (rightpt+leftpt)/2;  count++;  } while ((rightpt - leftpt) > epsilon);  //final output.  printf("\nApproximate solution = ");  printf(" %15.10lf\n", root);  return 0;  } |
|  |
| Finding the root for three degree polynomial Ax^3 + Bx^2 + Cx + D  using Bisection method.  Enter values for the following coefficients  A :1  B :3  C :-1  D :-4  Initial Bracket for root :  Low value:1  High value:2  1 :New interval: [ 1.0000000000 .. 1.5000000000 ]  2 :New interval: [ 1.0000000000 .. 1.2500000000 ]  3 :New interval: [ 1.0000000000 .. 1.1250000000 ]  4 :New interval: [ 1.0000000000 .. 1.0625000000 ]  5 :New interval: [ 1.0625000000 .. 1.0937500000 ]  6 :New interval: [ 1.0937500000 .. 1.1093750000 ]  7 :New interval: [ 1.1093750000 .. 1.1171875000 ]  8 :New interval: [ 1.1093750000 .. 1.1132812500 ]  9 :New interval: [ 1.1132812500 .. 1.1152343750 ]  10 :New interval: [ 1.1132812500 .. 1.1142578125 ]  11 :New interval: [ 1.1142578125 .. 1.1147460938 ]  12 :New interval: [ 1.1147460938 .. 1.1149902344 ]  13 :New interval: [ 1.1147460938 .. 1.1148681641 ]  14 :New interval: [ 1.1148681641 .. 1.1149291992 ]  15 :New interval: [ 1.1148681641 .. 1.1148986816 ]  16 :New interval: [ 1.1148986816 .. 1.1149139404 ]  17 :New interval: [ 1.1148986816 .. 1.1149063110 ]  18 :New interval: [ 1.1149063110 .. 1.1149101257 ]  19 :New interval: [ 1.1149063110 .. 1.1149082184 ]  20 :New interval: [ 1.1149063110 .. 1.1149072647 ]  21 :New interval: [ 1.1149072647 .. 1.1149077415 ]  22 :New interval: [ 1.1149072647 .. 1.1149075031 ]  23 :New interval: [ 1.1149075031 .. 1.1149076223 ]  24 :New interval: [ 1.1149075031 .. 1.1149075627 ]  25 :New interval: [ 1.1149075031 .. 1.1149075329 ]  26 :New interval: [ 1.1149075329 .. 1.1149075478 ]  27 :New interval: [ 1.1149075329 .. 1.1149075404 ]  28 :New interval: [ 1.1149075404 .. 1.1149075441 ]  29 :New interval: [ 1.1149075404 .. 1.1149075422 ]  30 :New interval: [ 1.1149075404 .. 1.1149075413 ]  31 :New interval: [ 1.1149075413 .. 1.1149075418 ]  32 :New interval: [ 1.1149075413 .. 1.1149075415 ]  33 :New interval: [ 1.1149075413 .. 1.1149075414 ]  34 :New interval: [ 1.1149075414 .. 1.1149075415 ]  Approximate solution = 1.1149075415  Process returned 0 (0x0) execution time : 31.573 s  Press any key to continue. |
| Homework : \_\_8C\_\_\_\_  //Christopher Badolato  //ENC 3211  //11/29/18  //Assignment 8  /\*  source code was borrowed from  https://www.codewithc.com/c-program-for-gauss-jordan-method/  all of the code from the website was exactly what i needed.  used the dummy matrix to first test.  \*/  #include<stdio.h>  int main(){  int i,j,k,n;  float A[20][20],c,x[10];  /\*  float dummyMatrix[3][3] = { 2.000,4.000,6.000,22.000,  3.000,8.000,5.000,27.000,  -1.000,1.000,2.000,2.000};  \*/  printf("\nEnter the size of matrix: ");  scanf("%d",&n);  printf("\nEnter the elements of augmented matrix row-wise:\n");  for(i=1; i<=n; i++)  {  for(j=1; j<=(n+1); j++)  {  printf(" A[%d][%d]:", i,j);  scanf("%f",&A[i][j]);  }  }  //Now finding the elements of diagonal matrix  for(j=1; j<=n; j++)  {  for(i=1; i<=n; i++)  {  if(i!=j)  {  c=A[i][j]/A[j][j];  for(k=1; k<=n+1; k++)  {  A[i][k]=A[i][k]-c\*A[j][k];  }  }  }  }  printf("\nThe solution is:\n");  for(i=1; i<=n; i++)  {  x[i]=A[i][n+1]/A[i][i];  printf("\n x%d=%f\n",i,x[i]);  }  return 0;  } |
|  |
| Enter the size of matrix: 3  Enter the elements of augmented matrix row-wise:  A[1][1]:2.000  A[1][2]:4.000  A[1][3]:6.000  A[1][4]:22.000  A[2][1]:3.000  A[2][2]:8.000  A[2][3]:5.000  A[2][4]:27.000  A[3][1]:-1.000  A[3][2]:1.000  A[3][3]:2.000  A[3][4]:2.000  The solution is:  x1=3.000000  x2=1.000000  x3=2.000000  Process returned 0 (0x0) execution time : 37.244 s  Press any key to continue. |